The Proposed Sticks Standard

by

Stephen Trimberger

Technical Report 3880

October 27, 1980

Computer Science Department

California Institute of Technology

Pasadena, California 91125

Silicon Structures Project

sponsored by

Burroughs Corporation, Digital Equipment Corporation,

Hewlett-Packard Company, Honeywell Incorporated,

International Business Machines Corporation,

Intel Corporation, Xerox Corporation,

and the National Science Foundation

Sticks Standard Proposal

This is version 1.0 of the Sticks Standard. Software has been written to interface this standard to plotters, a graphic Sticks editor, a Stick compactor and several simulators. The Standard appears adequate to describe cells for chip assemblers as well as Stick diagram editing and compaction systems. However, this version of the Sticks Standard cannot efficiently describe large chips because it lacks an array facility. This deficiency will be corrected in the next release of the Sticks Standard.

This document consists of four parts: Sticks definition, Sticks Standard design considerations, the specification of the Sticks Standard, and an example of the Standard in use.

Definition of Sticks

Sticks is a representation of integrated circuit data which incorporates physical and structural information [Williams 77]. There are three major pieces to a Sticks description of a circuit: a set of components, connectivity among components in the circuit, and a set of constraints on the positions of interesting features.

Sticks components may be electrical components. However, Sticks does not guarantee that there is a one-to-one relation between the electrically active objects in the Stick diagram and the electrically active objects in the circuit described by the Sticks.

Sticks is not intended to merely feed mask generation programs; the Sticks form can be used as input to a simulator, in which case the physical information is ignored. Sticks is also perfectly valid as a picture description, in which case the electrical information is ignored. Therefore, there is no constraint on the Sticks data that it be planar or that it even be a valid circuit.

Another important use of Stick diagrams is as "structural idiom", the LSI equivalent of an algorithm in software. The Stick diagram expresses a function and a general means for implementing that function, without tying the function tightly to a

particular technology or fabrication process, just as a good software algorithm is not tied to one programming language.

Sticks need not be limited to one integrated circuit technology or even to integrated circuit layout. This Sticks Standard reflects this view of Stick diagrams as an electronic design form. For that reason, the technology-dependent parts of Sticks are either relegated to the manipulating program altogether or tucked into small, well-defined corners of the Sticks Standard description.

The Sticks representation is close enough to mask geometry to allow easy translation to mask data. It is composed of electrical components, which are essential for simulation and performance estimation. It leaves unspecified the details of mask geometry and deals with logical connection, so the Sticks cells can be deformed by a chip assembler to fit better in a new design.

Design Considerations

The Sticks Standard is a data interchange form. It is expected that input will come from several different CAD systems. Therefore, the Standard cannot guarantee properties of the data not specified by the syntax. The relationship between features in the specification must be checked by a Sticks processing program.

The Sticks Standard has been designed primarily to transfer data between computers. The Standard is in a text form, which is independent of the type of machine and wordsize as well as the medium on which it is stored. Text is accepted by a wide variety of machines and text processing is usually handled very efficiently.

Because of the machine-to-machine nature of the Sticks Standard, human readability is secondary to machine readability. More importantly, machine-generated files are expected to be syntactically correct. Therefore, syntactic features such as error recoverability are secondary to parsing ease. This is contrary to much current thinking which is driven by years of programming language design.

The Sticks form is design-rule independent. Therefore a processing program, which is provided with detailed design rules, can be made as clever as desired. When the geometric design rules on a process line are improved, libraries need not be re-created, merely re-run through the updated compactor. The Stick diagram is the most specified form that can be passed to another process line for second-sourcing. The same Stick diagram can be passed through the compactor for the new line and the result will be a design rule correct version of the chip for the new process line.

A Sticks reader can possess detailed knowledge of the process, including detailed geometric design rules, whereas the writer can possess only generic knowledge of the process (NMOS, CMOS, etc.) and only functional knowledge of the circuit. It is the task of the reader to optimize the design for its process. This optimization may be concerned with minimum area, low power, high speed or some combination of these design parameters. Thus the reader performs a very large task, but there is only one reader needed per process line.

A Sticks Standard file should be able to represent an already-processed cell. The translation from that form of the cell to mask geometry would be trivial. With this capability, a Sticks compactor could both read and write Sticks Standard files. When reading Sticks Standard files, the compactor should ignore those parts dealing with physical locations.

The Sticks Standard file should allow references to cells defined elsewhere. Cells need not be fully represented in the Sticks form, but their interface should be easily made.

The Sticks Standard does not include the facilities to parameterize a cell. In order for parameters to be useful, the facility must exist to produce functions of the parameters. This leads to a need for arithmetic expressions and a function definition capability. Programming language constructs such as conditional expressions and looping expressions are needed to fully use the parameters. This complexity is not necessary to interchange Stick diagrams. A cell with parameters could alternatively be defined as several cells with the legal parameter sets encoded in the cell name.

Parameters are not needed in this interchange form. A procedure in a sequential programming language must be coded without foreknowledge of the values of the

parameters used. The Sticks Standard has a descriptive semantics, so that, conceptually, each cell can be constructed with the knowledge of the parameters to be used with it. Therefore, without loss of generality, cells may be defined with the parameter list encoded in the cell name. There will not be an impossibly large number of cells created in this fashion because the calls must be individually listed.

The Sticks Standard

The Sticks Standard has a descriptive rather than a procedural semantics. That is, it describes an image rather than the means for creating the image. The Standard is intended to be a means for data interchange, not a database. Many suggestions for extensions and changes can be traced to a conceptualization of the use of the Sticks Standard as a database.

A description in the Sticks Standard carries with it a set of coordinates for all interesting locations in the Sticks data. The locations are useful for plotting, and may be used by Sticks processing programs as an initial placement for compaction. In addition to the physical locations, the cell definition may contain a list of constraints on the final positions of the points in the design. The physical locations should be considered to be suggestive, whereas the constraints are imperative.

The definition of the syntax and semantics of the technology-dependent parts of the design are separate from this definition of the "pure" Sticks Standard, but the Sticks Standard cannot be used for a given electronics technology until those technology-dependent parts are defined. A set of technology-dependent definitions for NMOS is given with the example below.

This description of the Sticks Standard is separated into two parts: syntax and semantics.

Syntax

The syntax description consists of a formal Backus-Naur Form (BNF) description of the syntax and a discussion of some interesting features of that syntax.

Wirth's standard notation is used [Wirth 77]: production rules use equals to relate identifiers to expressions, vertical bar for choice and double quotes around terminal characters. Curly brackets indicate zero or more repetitions, square brackets indicate optional features and parentheses are used for grouping.

The formal syntax description is divided into two parts: one for token scanning, one for parsing. The BNF for token scanning is ambiguous when the scanner encounters a string of characters. According to the BNF, a space may appear as zero repetitions between characters in a name or number. This ambiguity is resolved in the obvious manner: in every case, the largest legal string is chosen as the next token.

BNF For Token Scanning

```
file                = space { token space }
token               = name | number | keyword | specialChar
keyword             = "CELL" | "MACRO" | "COMPONENTS" | "TWIGS" |
                      "CONSTRAINTS" | "TOP" | "LEFT" | "RIGHT" | "BOTTOM" |
                      "X" | "Y" | "POINT" | "CONNECTOR" | "END"
specialChar         = "(" | ")" | ";" | "." | ":" | "[" | "]" |
                      "<" | ">" | "=" | "-" | "+"
name                = letter { nameChar }
number              = [ "-" ] digit { digit }
letter              = "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"
nameChar            = letter | digit | "_"
digit               = "0" | "1" | "2" | ... | "9"

space               = { sepchar } | space "[" commentText "]" space
sepChar             = any character except nameChar, specialChar
commentText         = { commentChar } |
                      commentText "[" commentText "]" commentText.
commentChar         = any character except "[" or "]"
```

BNF For Parsing

```
file                = { celldef }
celldef             = CellHeader { celldef } ComponentDefinition
                      TwigDefinition ConstraintSpec "END"
CellHeader          = ( "CELL" | "MACRO" ) name number number
ComponentDefinition = "COMPONENTS" {ComponentType paramText ":"
                      compDecl { compDecl } ";" }
TwigDefinition      = "TWIGS" { colorName paramText ":" [ twigName ] "="
                      TwigEntry ";" }
ConstraintSpec      = "CONSTRAINTS" { ConstraintStmt ";" }

ComponentType       = name | "POINT" | "CONNECTOR"
compDecl            = compName [ orientation ] point
paramText           = { any character except "[" "]" ":" }

colorName           = name
compName            = name
connName            = name
twigName            = name

orientation         = ( "N" | "M" ) number number
TwigEntry           = TwigPrimitive | "(" TwigEntry ")" |
                      TwigPrimitive TwigEntry
TwigPrimitive       = compName [ "." connName ] | point

ConstraintStmt      = orderPrimitive | ConstraintStmt orderOp ConstraintStmt |
                      "(" ConstraintStmt ")"
orderPrimitive      = compName [ "." fieldname ] [ displacement ] |
                      orderKeyword | number
displacement        = ( "-" | "+" ) number
orderKeyword        = "LEFT" | "RIGHT" | "TOP" | "BOTTOM"
orderOp             = "<" | "=" | ">"
fieldname           = "X" | "Y"

point               = number number
```

The file contains a list of cell definitions. Each cell begins with either "CELL" or "MACRO" and ends with "END". The cell description is divided into four sections: Header, Component Definition, Twig Definition, and Constraints. Each section starts with a keyword. Cell definitions may be nested within other definitions.

All numbers are integers, and are considered to be in units of hundredths of a micron. The header of the cell has a scale factor to be applied to the numbers in the cell. Therefore, numbers in the cell could be given in lambda, with the scaling from lambda to hundredths of a micron given on the cell header. Comments can be included anywhere a space can be put and can be removed in the lexical scan.

Semantics

The Sticks file consists of components, interconnect, and constraints on the physical layout. The header gives the cell name and scale, the component definition indicates the type of each kind of component. The twig definition section describes

the twigs, their connections, and their paths. The constraints section specifies restrictions on the physical layout.

The points given with the components, twigs, and in constraint displacements give a sample physical arrangement of the components and twigs. If used for plotting, the units should be hundredths of a micron. The physical locations may be considered mere suggestions by a Sticks processing program.

The Sticks File

The Sticks file contains a list of Sticks cells. Therefore, two designs can be merged by concatenating the files if the names of the top-level cells in each of the designs are unique. If the names are not unique, one of the designs can be enclosed in a separate cell definition with a unique name.

The Header

The cell header consists of the name of the cell and a scaling factor. The scaling factor consists of two numbers: a and b. All numbers in the cell are scaled by a*number/b. The scaling is not applied to instances of other cells declared in the component list or to cells defined in the scope of the current cell. The scale allows the file to contain more meaningful numbers -- for example, lambda dimensionless units.

The header includes an indication of the type of the cell. "MACRO" cells are those which have already been compacted or have been defined elsewhere. These include hand-drawn circuitry as well as PLAs and ROMs produced from specialized generators. The precise description of these cells need not be included in the Sticks Standard file. The MACRO definition is used to specify the interface to those cells in a uniform manner. The Components, Twigs and Constraints sections specify the types, colors, and positions of the connectors.

Component Definition

Components are declared in a fashion similar to variable declaration in a high-level programming language. A component may be a point, a connector, a reference to a cell defined in the current context, or a technology-dependent name. Technology

dependent names denote transistors, resistors, contacts, and similar features in that technology. A sample component definition for NMOS technology is given with the example below.

Each component has with it a position and an optional orientation, which includes mirroring and rotation information. This orientation, like the physical locations on components, is mere suggestion, and may be altered by a Sticks processing program. The orientation consists of a one-letter indication of the mirroring of the coordinate system a "N"ormal coordinate system has the +y axis counterclockwise from the +x axis. A "M"irrored coordinate system has the component mirrored about the (1,1) vector. The two numbers following the mirroring key is the rotation key. The two numbers give the x-y coordinate of the direction to which to rotate the +x axis.

There are two predefined component types: points and connectors. Points are simply interesting locations in the design. Twigs may be routed through points to provide the initial topology of the design. Connectors are named locations to which connections to instances of the cell are made. Connectors may be parameterized, giving the connector a type which can subsequently be used by a chip assembler to maintain compatibility when cells are merged into systems. A simple connector typing scheme is given with the example below.

The Instantiation Hierarchy

Besides points, connectors, and technology-dependent components, a component may be a reference to another cell. Any cell defined in the current context may be referenced as a component. This gives rise to the instantiation hierarchy, sometimes known as the "calling" hierarchy. The hierarchy simplifies the design by reducing the amount of data necessary to specify the design.

Twig Definition

A Sticks twig is a connected path with a given color and set of parameters (for example, line width). The twig may or may not be a true electrical node, as it may run into a contact component which may make contact to the same layer or other layers. It may intersect other twigs. The electrical interaction at the crossing of twigs is not specified, and the Sticks Standard does not restrict the crossing in any way. Such a construct might violate design rules in a particular technology, and

could be checked. Twig attributes cannot be changed as they are specified only once for the twig.

A twig has optional data following the color name to allow specification of technology-dependent information. The legal color names and the parameters for a twig must be defined when the technology dependent parts of Sticks for a particular technology is announced. Sample twig color names and parameters for NMOS technology are given with the example below.

The twig definition describes the path taken by the twig. The path can have branches. The connectors on components are referenced by their names and the component name. Unnamed points may be specified directly in the twig definition using a coordinate pair. Two unnamed points at the same coordinate are assumed to be separate points and their identical positions mere co-incidence.

The path is routed through the points given in the TwigEntry to the components. Mere connection can be easily represented for those applications where paths are unnecessary or redundant. A branch in a twig is represented by a parenthesized point list. The TwigEntry in parentheses branches off the point given immediately before the parentheses.

The twig may be named if desired. The name is not used in the Sticks Standard, but may be useful later in viewing the Sticks or assembling the Sticks data into a form useful for simulation. There is no guarantee by the Sticks Standard that twigs with the same name all belong to the same electrical node.

Constraints

The constraints section gives a list of restrictions on the final positions of components in the design. The constraints should be met by the physical locations given in the Components and Twig Definition section, and Sticks processing programs could check this case.

There are three constraint operators for ordering: <, >, =. They are the familiar relational operators and force their ordering.

There are four keywords in the Constraint section: LEFT, RIGHT, TOP, BOTTOM. They refer to the four edges of the bounding box of the cell. It is implied that LEFT < everything < RIGHT and BOTTOM < everything < TOP. A point can be constrained to be equal to one of the keyword values, in which case it will always be positioned at the appropriate edge of the cell. The keywords need not be used in the description, but their values represent the physical limits of the cell and as such may be very helpful when setting connections to the outside world.

An optional displacement can be added to a constraint. This displacement can be used to set a limit to deformation offset from a component. In addition, an orderPrimitive may be a number which restricts the legal physical locations which a component may take. These numbers are in the same units as the others in the cell, and like those others are mere suggestion and are expected to be changed by Sticks processing programs.

Constraints can be applied to the x or y field on a component. This descriminates against those people laying out electronics in true three dimensional form, and this form of ordering might not adequately describe a design laid out in polar coordinates -- a rather minor point. In either case, additional field names for constraints or additional constraint operators could be created.

Circular constraints can be built, and the reader of the Sticks Standard file is advised to beware of them.

The Definition Hierarchy

Cells are defined within the context of a cell. This is the basis for the definition hierarchy, a hierarchy that limits the scope of defined cell names. A cell may only be used as a component inside its parent cell or inside all cells defined above that cell in the definition hierarchy. Two cells defined in same context cannot have the same name. However, cells defined in different contexts can have the same name with no confusion. In all cases, when reference is made to a cell, the nearest correctly-named cell in the definition hierarchy tree is used.

Technology-Dependent Parts

There are pieces of a Sticks specification that are not predefined by "pure" Sticks. This allows the framework of the Sticks Standard to be used for more than one integrated circuit technology. For a given technology, the component names and twig colors must be named, their parameter syntax and semantics must be given, and the geometrical representation of the components and twigs must be specified. The names may be made deliberately unique within an integrated circuit technology as well as across technologies to avoid confusion. Parameters to components can be anything that the processing programs can interpret: numbers, keywords, paths or arbitrary polygons. Parameters may or may not have precise geometrical or electrical meaning.

Component parameters and twig parameters are similar in their form and their loose definition. Component parameters give types of connectors, transistor ratios or similar information. Twig parameters identify wiring layer widths or capacity.

An Example

The Sticks Standard can be used with any technology. To do so, one must define the component names, the twig color names and their parameters. In addition, one must define the precise geometrical representation of the components and twigs. This geometric representation is not represented in the Sticks Standard file. It is similar in nature to design rule definition, as it deals with the geometrical layout of the cell. All such information is part of the Sticks processing system. The Sticks Standard file is independent of design rules so one file can be used with a number of different processing characteristics and design rules to optimize for different process lines.

This section contains a sample set of definitions for NMOS Sticks. Included in this definition is their representation in the Sticks file and the geometrical meaning of those components, including the geometrical layout and the location of connection points.

Components

All transistors have three connectors, named source, drain and gate. In all transistors, the diffusion and polysilicon overlap the edges of the active area by the amount specified by a design rule. All numbers are in the same units as other numbers in the cell.

NENH(width source-x source-y drain-x drain-y path) is an enhancement transistor with an integer width, source and drain connection points, and a list of points for the path of the gate. Points are x-y coordinate pairs as defined in the Sticks Standard. The numbers are displacements from the position of the gate connection point. Therefore, one of the points in the path should be 0,0.

NDEP(width path) is a depletion transistor with an integer width and a list of points for the path of the channel. Points are x-y coordinate pairs as defined in the Sticks Standard. The numbers are displacements from the gate connector of the transistor. The gate connection point is at the origin of the coordinates. The source and drain connection points are at the ends of the channel path.

NTRN(width length) and NDTR(width length) are rectangular enhancement and depletion transistors, respectively, with integers for the width and length of the channel. The gate runs horizontally. All connectors connect to the center of the gate area.

NRES(width length) is a rectangular depletion transistor (resistor) with two integers for width and length of the channel. The resistor has a butting contact embedded inside the channel making a connection between the source and gate. The diffusion runs vertically. The depletion transistor has two connectors, named in and out, both at the common diffusion-polysilicon area in the middle of the butting contact.

NCON is a contact between metal and any other layer. It has no connectors. Connections are made to the origin at the center of the contact.

NBUT is a butting contact between polysilicon and diffusion. Diffusion is left of polysilicon. NBUT has no connectors. Connections are made to the the center of the common diffusion-polysilicon area.

NBUR is a buried contact between polysilicon and diffusion. Diffusion is left of polysilicon. NBUR has no connectors. Connection is made to the center of the common diffusion-polysilicon area.

The CONNECTOR component parameters are "input", "output" or both.

Twigs

The twig color names are the layer names: Diffusion, Poly, Metal. The parameter on a twig is an optional width of the path in the same units as the other numbers in the cell. If no width is specified, the width defaults to the design rule minimum width.

References

[Mead 80] C.A. Mead and L.A. Conway, Introduction to VLSI Systems, Addison Wesley, 1980

[Williams 77] J.D. Williams, "Sticks -- A New Approach to LSI Design", Master's Thesis, MIT, 6/77

[Wirth 77] N. Wirth, "What Can We Do About the Unnecessary Diversity of Notations for Syntactic Definitions?", Communications of the ACM, 11/77

Example of the Sticks Standard in Action

The following is a shift register cell in the Sticks Standard form.

```
CELL SRcell 200 1  [ lambda = 2 microns ]

[ This is the usual demo -- the shift register cell. ]

COMPONENTS
    CONNECTOR (input): input 0,5;
    CONNECTOR (output): output 20,5;
    CONNECTOR (input output): gndin 0, 0, gndout 20,0, pwrin 0,20,
    pwrout 20,20, clktop 15,20, clkbottom 15,0;
    NTRN (4 2): pd N 1 0 5,5;
    NRES (2 4): pu 5,15;
    NCON: gc 5,0, pc 5,20;
    NBUT: tored 17,5;
    POINT:. downbend 10,10;

TWIGS
    Metal: GND = gndin gc gndout;
    Diffusion (4): = gc pd.source;
    Metal: PWR = pwrin pc pwrout;
    Diffusion: = pc pu.in;
    Poly: IN  =  input pd.gate;
    Poly: OUT = tored output;
    Diffusion: = pu.out 5,10 (pd.drain) downbend 10,5 tored;
    Poly: = clktop clkbottom;

CONSTRAINTS
    LEFT = gndin.X = pwrin.X = input.X;
    gndin.X < (pc.X = pu.X = pd.X = gc.X) < downbend.X;
    clktop.X < tored.X;
    clkbottom.X < tored.X <
        output.X = gndout.X = pwrout.X = RIGHT;

    BOTTOM = gndin.Y = gndout.Y = gc.Y = clkbottom.Y <
        (input.Y = output.Y) <
        downbend.Y < (pwrin.Y = pwrout.Y = pc.Y = clktop.Y = TOP);
    pd.Y < pu.Y;
END
```

Clever Picture of the Example